REPAST STATECHARTS GUIDE

JONATHAN OZIK, NICK COLLIER - REPAST DEVELOPMENT TEAM

0. Before we Get Started

Before we can do anything with Repast Simphony, we need to make sure that we have a proper installation of Repast Simphony 2.1. Instructions on downloading and installing Repast Simphony on various platforms can be found on the Repast website.

1. Getting Started with Statecharts

1.1. Adding Statecharts.

A statechart can be added to any Java, Groovy or ReLogo class. Right click the class and select New → Statechart Diagram (Figure 1.

1.2. Statecharts Workspace.

2. States

One of the fundamental building blocks of statecharts are states. Here we introduce the different types of states that existing within the Repast Simphony statecharts framework.

2.1. Entry State Marker.

Every statechart must have an entry state marker. This defines the path through which the statechart begins when it is activated.

Date: June 26, 2013.

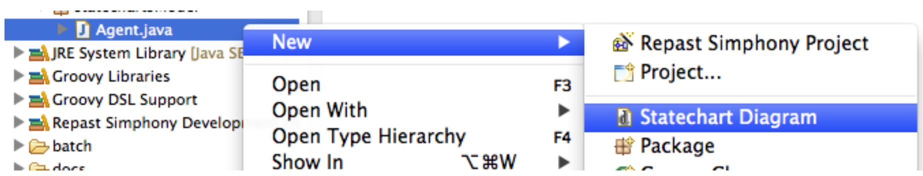


FIGURE 1. Creating a new statechart.

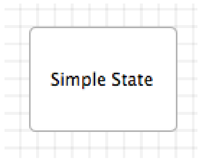


FIGURE 2. Simple state.

2.2. Simple State.

A simple state looks like Figure 2. At any one point in time within an active statechart, one and only one of the simple states will be active. In addition to their *ID*, simple states can have *On Enter* and *On Exit* actions defined, as seen in the simple state properties panel in Figure 3. These actions are triggered when entering or exiting the simple state, respectively. The keywords available within the two action blocks are:

agent: This is the agent that contains the statechart. Any method (e.g., `customMethod`) defined on the agent can be invoked through this reference (e.g., `agent.customMethod()`).

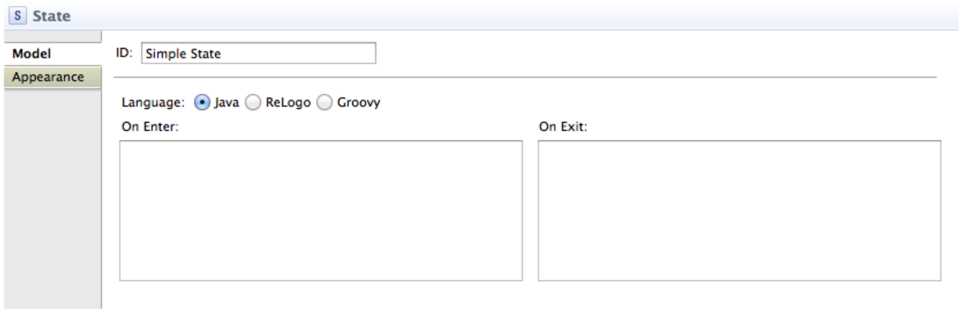
state: This is the state itself. For example, the state's *ID* can be accessed via: `state.getId()`.

params: This is the model's `Parameters` object. As an example, a double valued parameter "dParam" can be retrieved with: `params.getDouble("dParam")`[1].

As is the case with all types of action blocks, their logic can be specified using Java, Groovy or ReLogo. Specifically, any Java, Groovy or ReLogo code can be used to express the type of behavior that should be executed upon entry to or exit from the state[2].

[1]See the source or JavaDoc for `repast.simphony.parameter.Parameters` for all of the available methods.

[2]When using the ReLogo option, the `agent` parameter is implicit so writing `customMethod()` is equivalent to `agent.customMethod()`.


State

Model

Appearance

ID: Simple State

Language: [✓ Java] ReLogo Groovy

On Enter:

On Exit:

Figure 3. Simple state properties.

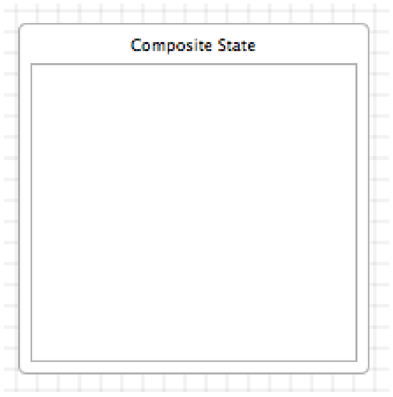
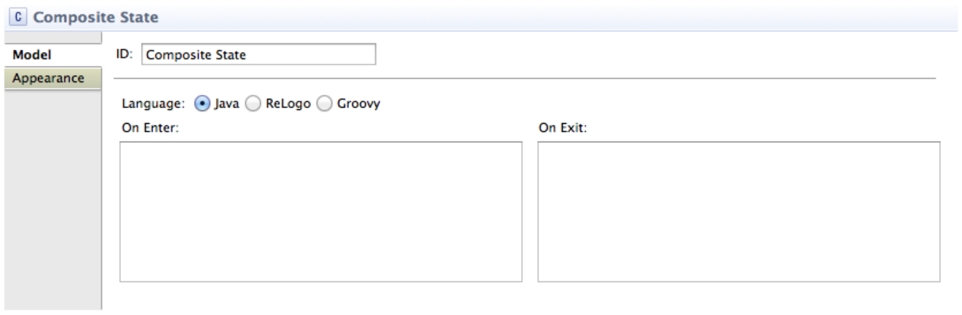


FIGURE 4. Composite state.



ID: Composite State

Language: [✓ Java] ReLogo Groovy

On Enter:

On Exit:

FIGURE 5. Composite state properties.

2.3. **Composite State.**

Composite states are used to nest elements within a statechart. Figure 4 shows an empty composite state. Composite states can include the following elements:

- Simple state (Section 2.2)
- Composite state (Section 2.3)
- Initial state marker (Section 2.4)
- History state (Section 2.5)
- Final state (Section 2.6)
- Branching state (Section 2.7)

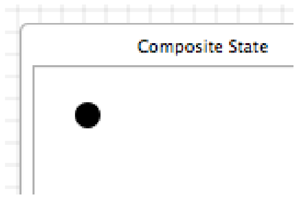


FIGURE 6. Initial state marker (within a composite state).

2.4. **Initial State Marker.** ●

2.5. **History State.**

2.6. **Final State.**

2.7. **Branching State.** ◇

3. TRANSITIONS

Transitions between states make up the other type of fundamental building block of statecharts. In this section we introduce the different types of transitions that can be used within the Repast Simphony statecharts framework.

There are two overall types of transitions, *regular transitions* (Figure 7) which connect different states and *self transitions* (Figure 8) which are internal to a state[3]. For any type of transition an *On Transition* action can be defined. This action will be executed whenever the transition is traversed. The keywords available within the *On Transition* action block are:

agent: This is the agent that contains the statechart.
transition: This is the transition itself. For example, the transition's source state can be accessed via: `transition.getSource()`[4].
params: This is the model's `Parameters` object.

For almost all types of transitions[5] a *Guard* condition can be defined. A *Guard* condition is an additional boolean condition that has to be satisfied for a transition that is valid to be actually considered as a candidate for traversal. This condition is specified by a block of code that returns a boolean and the keywords available in a *Guard* condition are the same as those in an *On Transition* action block.

[3]One also has the ability to define a *regular transition* that begins and ends at the same state. Unlike the *self transition* case, each time the *regular transition* is taken, the state will be exited and subsequently re-entered.

[4]See the source or JavaDoc for repast.simphony.statecharts.Transition for all of the available methods.

[5]All transitions except default transitions out of branching states.

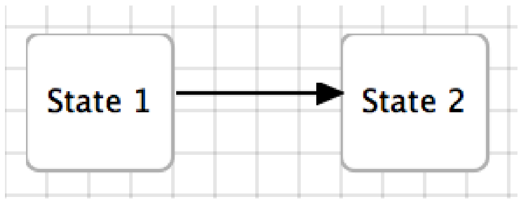


FIGURE 7. Regular transition between State 1 and State 2.

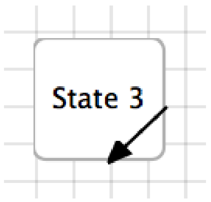
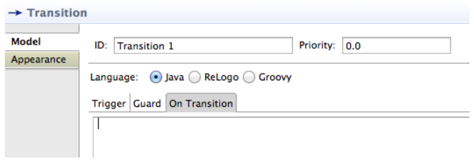


FIGURE 8. Self transition internal to State 3.

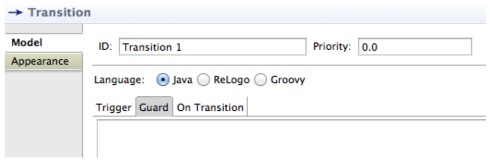


ID: Transition 1 Priority: 0.0

Language: [✓ Java] ReLogo Groovy

Trigger Guard [✓ On Transition]

FIGURE 9. Properties for a transition showing the *On Transition* action block.



ID: Transition 1 Priority: 0.0

Language: [✓ Java] ReLogo Groovy

Trigger [✓ Guard] On Transition

FIGURE 10. Properties for a transition showing the *Guard* condition block.

3.1. **Always Transition.**

3.2. **Timed Transition.**

3.3. **Probability Transition.**

3.4. **Condition Transition.**

3.5. **Exponential Decay Rate Transition.**

3.6. **Message Transition.**

3.6.1. *When Message Meets Condition.*

3.6.2. *When Message Equals.*

3.6.3. *When Message is of Class.*

3.6.4. *Always.*